Design of a Multi-Cycle Processor

Sri Sai Nomula
SR No: 22986, MTech ESE,

I OBJECTIVE

The Processor is a 16-bit RISC CPU with 16-bit Address space with RISC type instructions. The processor is a multi-cycle processor which does fetch, decode, execution, and write stages sequentially in multiple cycles. The processor has an Instruction prefetch register (IPR) available, the output of which goes to instruction register (IR) so that instruction fetch of the next instruction can happen concurrently with the processing of the current instruction.

II SPECIFICATIONS

- 16-bit Data width and 16-bit Instruction width.
- 8 General Purpose Registers- R0 to R7 (16 bit each).
- A Instruction pre-fetch register to fetch next instruction while current instruction is executed.
- Instructions supported- ADD, SUB, AND, OR, LOAD, STORE, BGE, BEQ, JUMP, ADDI and HLT.

III INSTRUCTION FORMAT

The first 3 bits of the instruction are used to define the instruction, and the rest of bits are used to define the function or immediate values and Register addresses. The table below shows how each instruction is divided.

Instruction	[15:12]	[11:9]	[8:6]	[5:3]	[2:0]
R-Type	Function	RS2	RS1	RD	OpCode
Load	Imm(6:3)	Imm(2:0)	RS1	RD	OpCode
Store	Imm(6:3)	Imm(2:0)	RS1	RD	OpCode
ADDI	Imm(6:3)	Imm(2:0)	RS1	RD	OPCode
BEQ	Imm(6:3)	RS2	RS1	Imm(2:0)	OpCode
BGE	Imm(6:3)	RS2	RS1	Imm(2:0)	OpCode
Jump	Imm(6:3)	XXX	XXX	Imm(2:0)	OpCode
HLT	XXXX	XXX	XXX	XXX	OpCode

Table 1: Instruction Set Encoding

1. **R-Type Instruction:** The OpCode is 000 and the Funct field describes the operation ALU should perform (0000 - ADD, 0001 - SUB, 0010 - AND, 0011 - OR). RS1, RS2 and RD fields provides the Register Values which are of 3-bits each. Example - ADD R1 R2 R3 [R1 <- R2 + R3]. Here, Funct = ADD(0000),RS2 = 011 , RS1 = 010 and RD = 001.
2. **Load-Type Instruction:** The OpCode is 001 and the Imm field gives the Value which needs to added to RS1 and access that particular memory location. RS1 and RD fields provides the Register Values which are of 3-bits each. Example - LOAD R2 R1 101 [R2 <- mem[R1 + 101]]. Here, Imm = 101, RS1 = 001 and RD = 010.
3. **Store-Type Instruction:** The OpCode is 010 and the Imm field gives the Value which needs to added to RS1 and store into that particular memory location. RS1 and RD fields provides the Register Values which are of 3-bits each. Example - STORE R2 R1 101 [R2 -> mem[R1 + 101]]. Here, Imm = 101, RS1 = 001 and RD = 010.
4. **ADDI-Type Instruction:** The OpCode is 011 and the Imm field gives the Value which needs to added to RS1 and store into RD. RS1 and RD fields provides the Register Values which are of 3-bits each. Example - ADDI R2 R1 101 [R2 <- R1 + 101]. Here, Imm = 101, RS1 = 001 and RD = 010.

5. **BEQ(Branch on Equal)-Type Instruction:** The OpCode is 100 and the Imm field gives the Value where PC needs to points in case condition satisfies. RS1 and RS2 fields provides the Register Values which are of 3-bits each. Example - BEQ R3 R4 1101 [If R3=R4 then Fetch Instruction from Memory Location 1101]. Here, Imm = 1101, RS1 = 011 and RS2 = 100.

6. **BGE(Branch on Greater)-Type Instruction:** The OpCode is 101 and the Imm field gives the Value where PC needs to points in case condition satisfies. RS1 and RS2 fields provides the Register Values which are of 3-bits each. Example - BGE R3 R4 1101 [If R3 ¿ R4 then Fetch Instruction from Memory Location 1101]. Here, Imm = 1101, RS1 = 011 and RS2 = 100.

7. **JUMP-Type Instruction:** The OpCode is 110 and the Imm field gives the Value where PC needs to go. Example - JUMP 1101 [Fetch Instruction from Memory Location 1101]. Here, Imm = 1101.

8. **HALT-Type Instruction:** The OpCode is 111. This Instruction is used to END the Program. Example - HLT [Stop the Calculation]

IV DATA PATH AND CONTROL PATH

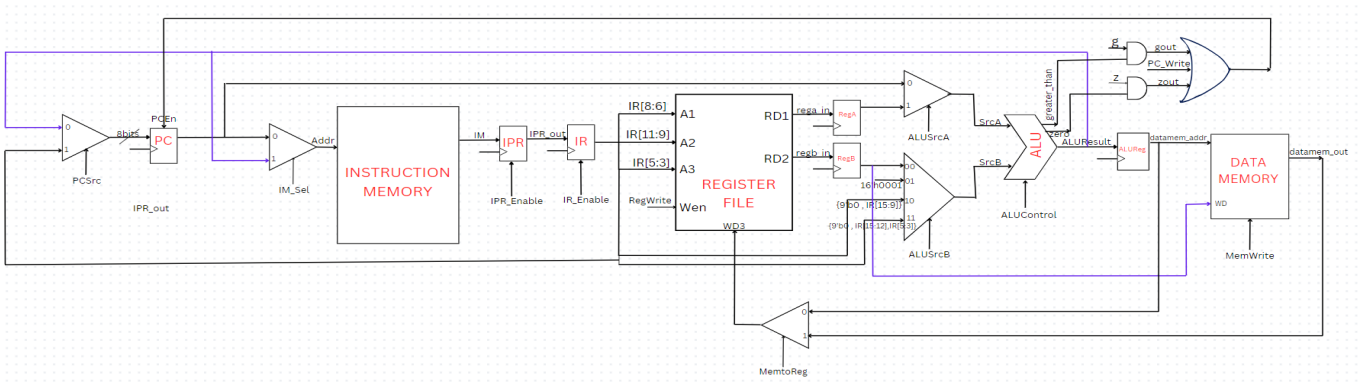

Figure 1: Level-1 Data Path Block Diagram

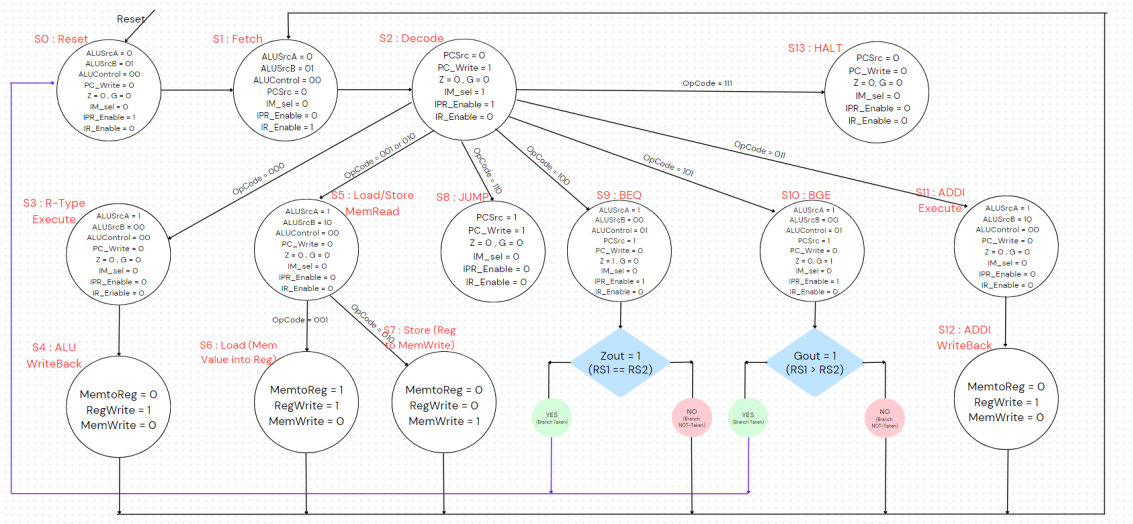

Figure 2: Level-1 Control Path State Diagram

The Functionality of various control signals are described in the Table Below.The values of these signals change based on the OpCode and the Task they are performing described by the Controller.

Signal	Function
ALUSrcA	To Select Operand-1 to ALU between PC Value and Register Value.
ALUSrcB	To Select Operand-2 to ALU between Register Value, 16'h0001 , Immediate Value for ADDI and Immediate Value for Branch Instructions.
ALUControl	To Perform the desired operation of ADD, SUB, AND, OR.
PCSrc	To Select the PCNext Value between PC + 1 and Immediate Address specified.
PCWrite	To load the next PC Value into the PC Register.
Z	Used while performing Branch Equal Operation. So, that when Branch is Equal the Zout is set.
G	Used while performing Branch Greater than Operation. So, that when Branch is Greater then Gout is set.
IMSel	The Select the Instruction Addr between PC and PC + 1. So that we can load the correct Instruction into Prefetch Register.
IPREnable	To load the Fetched Instruction into PreFetch Register.
IREnable	To load the Instruction from PreFetch Register into Instruction Register.
MemtoReg	To Select the data to be written into Register File between ALUResult and Data from Memory.
RegWrite	To Write the data into Register File
MemWrite	To Write the data into Data Memory.

Table 2: Description of Control Signals

V ASSEMBLY PROGRAM

The below assembly program is used to find the maximum number from a set of ten random numbers.This assembly code is written using the instructions declared for this 16-bit RISC-V architecture. The instructions are stored in the first 9 locations of the Instruction Memory and the 10 random numbers are stored in the 10 locations of Data Memory and the final result(Maximum Number) is stored in the R3 of Register File.

Address	Instruction	Representation[HEX Format]
0000	ADDI R1 R0 1	020B
0001	ADDI R2 R0 10	1413
0002	LOAD R3 R0 0	0019
LOOP : 0003	BEQ R1 R2 STOP	144C
0004	LOAD R4 R1 0	0061
0005	ADDI R1 R1 1	024B
0006	BGE R3 R4 LOOP	08DD
0007	ADD R3 R4 R0	0118
0008	JUMP LOOP	001E
STOP : 0009	HALT	0007

Table 3: Assembly Program

VI TIMING REPORT

The Worst Negative Slack (WNS) is 2.332ns at an operating Time Period of 10ns. Therefore,the Maximum Operating Frequency can be calculated as:

$$f_{max} = \frac{1}{10ns - 2.332ns} = \frac{1}{7.668} = 130.412MHz. \quad (1)$$

The Timing Report is shown in Fig.3

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.332 ns	Worst Hold Slack (WHS):	0.090 ns	Worst Pulse Width Slack (WPWS):	3.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	890	Total Number of Endpoints:	890	Total Number of Endpoints:	273

All user specified timing constraints are met.

Figure 3: Timing Report for Multi-Cycle Processor

VII SIMULATION RESULTS

The Waveform after Simulation is shown in Fig.4

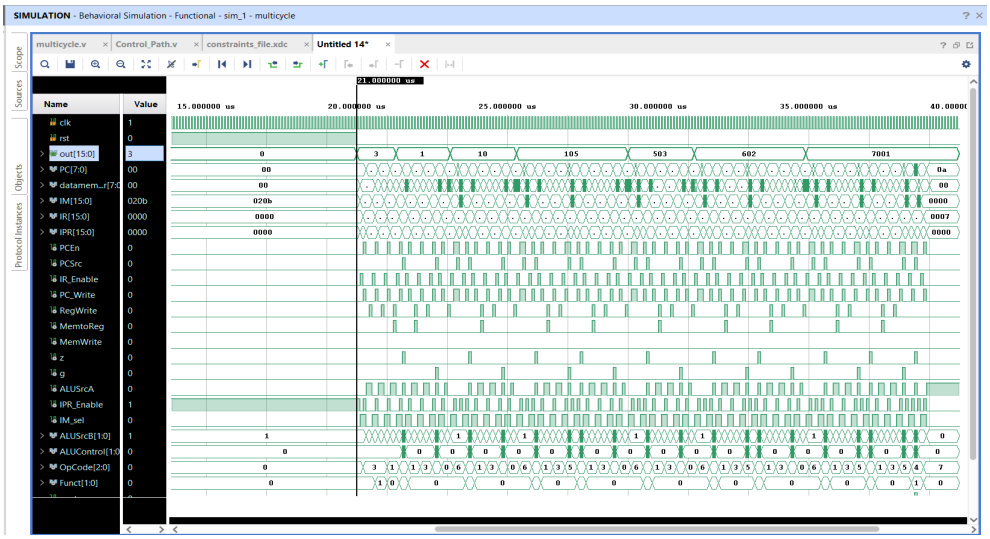

Figure 4: Simulation for Multi-Cycle Processor

VIII RESOURCE UTILIZATION

The component statistics of the Multi-Cycle Processor designed using Verilog Code gets synthesized into the Hardware as shown below in Fig.5 and Fig.6

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Bonded IOB (106)	BUFGCTRL (32)
multicycle	226	236	64	16	95	162	64	18	1
controller (Control_Path)	32	42	0	0	18	32	0	0	0
datapath (Data_Path)	197	194	64	16	90	133	64	0	0
ALU (ALU)	2	0	0	0	6	2	0	0	0
ALU_reg (flipflops_16)	0	16	0	0	4	0	0	0	0
datamem (Data_Memo	64	0	32	16	16	0	64	0	0
instrmem (Instruction_M	9	0	0	0	3	9	0	0	0
IPR (flipflops_16_0)	0	13	0	0	3	0	0	0	0
IR (flipflops_16_1)	22	13	0	0	14	22	0	0	0
PCreg (flipflops_8)	5	8	0	0	5	5	0	0	0
reg_file (Register_File)	72	112	32	0	53	72	0	0	0
reg_mux (mux2_16)	8	0	0	0	5	8	0	0	0
regA (flipflops_16_2)	15	16	0	0	23	15	0	0	0
regB (flipflops_16_3)	1	16	0	0	17	1	0	0	0

Figure 5: Components statistics of Multi-Cycle Processor

Summary

Resource	Utilization	Available	Utilization %
LUT	226	20800	1.09
LUTRAM	64	9600	0.67
FF	236	41600	0.57
IO	18	106	16.98

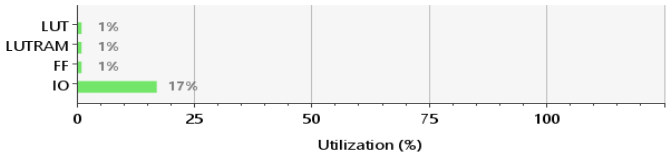

Figure 6: Components Summary of Multi-Cycle Processor

IX CONCLUSION

The designed 16-bit RISC Multicycle CPU was then verified using the assembly code for finding the maximum value in a set of 10 random Numbers and output was displayed on Basys3 FPGA Board using its 7-Segment display.

REFERENCES

[1] ”Digital Design and Computer Architecture 2nd edition” - David Mooney Harris and Sarah L Harris